Variable	Description
`$tcpinfo_rtt`, `$tcpinfo_rttvar`, `$tcpinfo_snd_cwnd`, `$tcpinfo_rcv_space`	If your operating system supports the `TCP_INFO` socket option, these variables will be populated with information on the current client TCP connection.
`$time_iso8601`, `$time_local`	Provides the current time in ISO 8601 and local formats respectively for use with the `access_log` directive.
`$uri`	Identical to `$document_uri`.

The location block

We have established that Nginx offers you the possibility to fine-tune your configuration down to three levels—at the *protocol* level (`http` block), the server level (`server` block), and the requested URI level (`location` block). Let us now detail the latter.

Location modifier

Nginx allows you to define `location` blocks by specifying a pattern that will be matched against the requested document URI.

```
server {
    server_name website.com;
    location /admin/ {
    # The configuration you place here only applies to
    # http://website.com/admin/
    }
}
```

Instead of a simple folder name, you can indeed insert complex patterns. The syntax of the `location` block is:

```
location [=|~|~*|^~|@] pattern { ... }
```

The first optional argument is a symbol called **location modifier** that defines the way Nginx matches the specified pattern, and also defines the very nature of the pattern (simple string or regular expression). The following paragraphs detail the different modifiers and their behavior.

The = modifier

The requested document URI must match the specified pattern exactly. The pattern here is limited to a simple literal string; you cannot use a regular expression:

```
server {
    server_name website.com;
    location = /abcd {
    […]
    }
}
```

The configuration in the `location` block:

- Applies to `http://website.com/abcd` (exact match)
- May apply to `http://website.com/ABCD` (it is only case-sensitive if your operating system uses a case-sensitive filesystem)
- Applies to `http://website.com/abcd?param1&param2` (regardless of the query string arguments)
- Does not apply to `http://website.com/abcd/` (trailing slash)
- Does not apply to `http://website.com/abcde` (extra characters after the specified pattern)

No modifier

The requested document URI must begin with the specified pattern. You may not use regular expressions:

```
server {
    server_name website.com;
    location /abcd {
    […]
    }
}
```

The configuration in the `location` block:

- Applies to `http://website.com/abcd` (exact match)
- May apply to `http://website.com/ABCD` (it is only case-sensitive if your operating system uses a case-sensitive filesystem)
- Applies to `http://website.com/abcd?param1&param2` (regardless of the query string arguments)

- Applies to `http://website.com/abcd/` (trailing slash)
- Applies to `http://website.com/abcde` (extra characters after the specified pattern)

The ~ modifier

The requested URI must be a case-sensitive match to the specified regular expression:

```
server {
    server_name website.com;
    location ~ ^/abcd$ {
    [...]
    }
}
```

The `^/abcd$` regular expression used in this example specifies that the pattern must begin (`^`) with `/`, be followed by `abc`, and finish (`$`) with `d`. Consequently, the configuration in the `location` block:

- Applies to `http://website.com/abcd` (exact match)
- Does not apply to `http://website.com/ABCD` (case-sensitive)
- Applies to `http://website.com/abcd?param1&param2` (regardless of the query string arguments)
- Does not apply to `http://website.com/abcd/` (trailing slash) due to the specified regular expression
- Does not apply to `http://website.com/abcde` (extra characters) due to the specified regular expression

With operating systems such as Microsoft Windows, `~` and `~*` are both case-insensitive, as the OS uses a case-insensitive filesystem.

The ~* modifier

The requested URI must be a case-insensitive match to the specified regular expression:

```
server {
    server_name website.com;
    location ~* ^/abcd$ {
    [...]
    }
}
```

The regular expression used in the example is similar to the previous one. Consequently, the configuration in the `location` block:

- Applies to `http://website.com/abcd` (exact match)
- Applies to `http://website.com/ABCD` (case-insensitive)
- Applies to `http://website.com/abcd?param1&param2` (regardless of the query string arguments)
- Does not apply to `http://website.com/abcd/` (trailing slash) due to the specified regular expression
- Does not apply to `http://website.com/abcde` (extra characters) due to the specified regular expression

The ^~ modifier

Similar to the no-symbol behavior, the location URI must begin with the specified pattern. The difference is that if the pattern is matched, Nginx stops searching for other patterns (read the following section about search order and priority).

The @ modifier

Defines a named location block. These blocks cannot be accessed by the client but only by internal requests generated by other directives such as `try_files` or `error_page`.

Search order and priority

Since it's possible to define multiple `location` blocks with different patterns, you need to understand that when Nginx receives a request, it searches for the `location` block that best matches the requested URI:

```
server {
    server_name website.com;
    location /files/ {
    # applies to any request starting with "/files/"
    # for example /files/doc.txt, /files/, /files/temp/
    }
    location = /files/ {
    # applies to the exact request to "/files/"
    # and as such does not apply to /files/doc.txt
    # but only /files/
    }
}
```

When a client visits `http://website.com/files/doc.txt`, the first `location` block applies. However, when they visit `http://website.com/files/`, the second block applies (even though the first one matches), because it has priority over the first one (it is an exact match).

The order you established in the configuration file (placing the `/files/` block before the `= /files/` block) is irrelevant. Nginx will search for matching patterns in a specific order:

- `location` blocks with the `=` modifier: If the specified string exactly matches the requested URI, Nginx retains the `location` block.
- `location` blocks with no modifier: If the specified string *exactly* matches the requested URI, Nginx retains the `location` block.
- `location` blocks with the `^~` modifier: If the specified string matches the beginning of the requested URI, Nginx retains the `location` block.
- `location` blocks with `~` or `~*` modifier: If the regular expression matches the requested URI, Nginx retains the `location` block.
- `location` blocks with no modifier: If the specified string matches the *beginning* of the requested URI, Nginx retains the `location` block.

In that context, the `^~` modifier begins to make sense, and we can envision cases where it would be useful.

Case 1

Let's begin by a simple example:

```
server {
    server_name website.com;
    location /doc {
    […] # requests beginning with "/doc"
    }
    location ~* ^/document$ {
    […] # requests exactly matching "/document"
    }
}
```

You might wonder: when a client requests `http://website.com/document`, which of these two `location` blocks applies? Indeed, both blocks match this request. Again, the answer does not lie in the order in which the blocks appear in the configuration files. In this case, the second `location` block will apply as the `~*` modifier has priority over the other.

Case 2

We will now take a look at this second example:

```
server {
    server_name website.com;
    location /document {
    […] # requests beginning with "/document"
    }
    location ~* ^/document$ {
    […] # requests exactly matching "/document"
    }
}
```

The question remains the same—what happens when a client sends a request to download `http://website.com/document`? There is a trick here. The string specified in the first block now exactly matches the requested URI. As a result, Nginx prefers it over the regular expression.

Case 3

Finally, let's go over a third and last scenario:

```
server {
    server_name website.com;
    location ^~ /doc {
    […] # requests beginning with "/doc"
    }
    location ~* ^/document$ {
    […] # requests exactly matching "/document"
    }
}
```

This last case makes use of the `^~` modifier. Which block applies when a client visits `http://website.com/document`? The answer is: the first block. The reason being that `^~` has priority over `~*`. As a result, any request with a URI beginning with `/doc` will be affected to the first block, even if the request URI matches the regular expression defined in the second block.